BAB IV
HASIL PENELITIAN DAN PEMBAHASAN

4.1. Pengumpulan Dataset

Dalam penelitian kali ini, digunakan beberapa dataset yang digunakan dengan tujuan yang berbeda-beda. Dataset yang digunakan antara lain adalah dataset COCO, Mall Dataset, MOT-20, dan dataset pribadi. Ada pun masing-masing dataset digunakan dengan tujuan masing-masing seperti berikut.

1. COCO (*Common Objects in Context*) Dataset, merupakan dataset yang menampung banyak gambar objek dan secara khusus digunakan untuk model-model objek deteksi. Dataset COCO ini juga biasa digunakan sebagai data untuk membandingkan performa antar model-model yang berkenaan dengan deteksi objek. Dalam penelitian kali ini, digunakan versi ringan dari dataset COCO, yakni COCO128. COCO128 merupakan dataset yang terdiri dari 128 gambar pertama pada *train-set* dataset COCO 2017.
2. *Mall* Dataset, merupakan suatu set data yang dikoleksi dari *webcam* publik dan umumnya digunakan untuk menghitung jumlah kerumunan (*crowd counting*). Dataset ini terdiri dari 2000 gambar yang diambil dari 2000 *frame* video.
3. MOT20 Dataset, merupakan suatu dataset yang bertujuan untuk pengujian ataupun testing algoritma pelacak (*tracker*). *Multiple-object Tracking* 20 (MOT20) dataset terdiri dari 8 video yang terbagi menjadi 2 set bagian, 4 *train-set* dan 4 *test-set*. Anotasi dataset ini sudah memiliki tingkat akurasi yang sangat tinggi dan dengan protokol yang sangat ketat. Pada penelitian kali ini, digunakan *train-set* sebagai dataset untuk *benchmarking* algoritma pelacak, yakni ByteTrack dan BoT-SORT.
4. Dataset pribadi, merupakan dataset yang diambil dengan cara direkam sebagai pengujian performa sistem secara kasus nyata (*real case*).

4.2. Pemrosesan Dataset

Hal berikutnya setelah tahap mengumpulkan dataset adalah memproses dataset. Pemrosesan terdiri dari proses *preprocessing* dan augmentasi. Berikut ini

adalah penjelasan dari masing-masing proses *preprocessing* dan augmentasi dataset.

4.2.1. *Preprocessing* Dataset

Proses *preprocessing* adalah suatu proses pengolahan data yang dilakukan untuk mempersiapkan dataset agar sesuai dan juga layak untuk dipakai untuk proses training model. Proses *preprocessing* pada penelitian kali ini meliputi *resizing*, *grayscaling*, dan *contrast stretching*. Berikut adalah beberapa proses *preprocessing* yang dilakukan pada penelitian kali ini.

1. *Resizing*, merupakan proses pengubahan ukuran panjang atau lebar gambar. Ukuran umumnya dalam *resizing* untuk model YOLO adalah 640 x 640 px. Oleh sebab itu, pada penelitian kali ini, dilakukan perubahan ukuran (*resize*) gambar-gambar pada dataset menjadi 640 x 640 px.
2. *Contrast Stretching*, merupakan proses peningkatan kontras pada gambar. Proses ini dilakukan untuk memperjelas bagian-bagian antar pixel, dengan cara meningkatkan perbedaan intensitas nilai antar pixel, sehingga ketajaman objek semakin terlihat jelas dan mudah diproses oleh model. Pada penelitian ini, dilakukan proses penajaman kontras dengan metode *Contrast Stretching*, yang meningkatkan kontras setiap pixel dengan meningkatkan rentang nilai intensitasnya.

4.2.2. Augmentasi Dataset

Proses augmentasi merupakan proses mendiversifikasi gambar-gambar dalam dataset, sehingga jumlah gambar dalam dataset bertambah dengan modifikasi berbagai macam yang digunakan untuk membuat model agar mampu menggeneralisir gambar dan meningkatkan performa model pada gambar yang tidak ada di dataset sebelumnya, proses augmentasi dalam penelitian kali ini meliputi *flipping*, 90° *rotation*, *rotation*, *shear*, *saturation*, dan *brightness*.

1. *Horizontal flip* dan *vertical flip*, merupakan proses membalikkan gambar baik secara horizontal atau pun vertikal. Proses ini dilakukan secara otomatis dengan kemungkinan membalikkan gambar secara horizontal dan

vertikal sebesar 50%. Berikut ini adalah hasil dari augmentasi *flip* seperti yang ditampilkan pada Gambar 4.1.

Gambar 4. 1 Gambar Hasil Augmentasi *Flip*

Berdasarkan Gambar 4.1, didapati bahwa *input* gambar dilakukan proses *flip* baik secara horizontal dan vertikal yang dengan probabilitas masing-masing adalah 50%.

2. 90° *Rotation*, merupakan proses rotasi sebesar 90 derajat (90°). Proses ini dilakukan secara otomatis untuk 3 kemungkinan, yakni tidak dirotasi, rotasi searah jarum jam (*clockwise*), dan rotasi berlawanan arah jarum jam (*anti-clockwise*). Masing-masing kemungkinan tersebut memiliki kemungkinan yang sama besar, sehingga setiap kemungkinan memiliki probabilitas terjadi sebesar 33,33%. Berikut ini adalah hasil dari augmentasi rotasi 90° seperti yang ditampilkan pada Gambar 4.2.

Gambar 4. 2 Gambar Hasil Augmentasi Rotasi 90 Derajat

Berdasarkan Gambar 4.2, didapati bahwa dataset akan secara otomatis memperkaya gambar-gambar dengan augmentasi 90°, masing-masing memiliki probabilitas 33.33%.

3. *Saturation*, proses augmentasi pada saturasi merupakan proses augmentasi yang memperkaya dataset dengan mengubah saturasi warna pada gambar-gambar. Saturasi merupakan tingkat kejernihan warna suatu gambar, sehingga model akan diberi dengan beberapa *input* gambar yang memiliki tingkat kejernihan warna yang berbeda-beda, semakin tinggi nilai saturasi pada gambar dataset, maka akan semakin jernih gambar pada dataset. Maka dari itu, pada proses augmentasi ini dilakukan augmentasi saturasi dari rentang -15% dan 15%, nilai augmentasi saturasi ini tidak terlalu rendah atau tinggi, sehingga tidak membuat model kesulitan mendeteksi pada saat proses *training* berlangsung. Berikut ini adalah hasil dari augmentasi saturasi seperti yang tertera pada Gambar 4.3.

Gambar 4. 3 Gambar Hasil Augmentasi Saturasi

Berdasarkan Gambar 4.3, dataset diperkaya dengan augmentasi saturasi, yakni tingkat kejernihan warna dari gambar. Hasil augmentasi saturasi ini terdistribusi secara merata dari rentang -15% hingga 15%.

4. *Brightness*, merupakan proses meningkatkan atau menurunkan level kecerahan pada gambar. Proses ini dilakukan agar model dapat meningkatkan performanya pada saat mendeteksi objek dengan kecerahan yang berbeda dengan gambar asli dalam dataset, sehingga model dapat mendeteksi pada pencahayaan yang lebih gelap maupun lebih terang dari tingkat kecerahan aslinya. Pada penelitian kali ini, augmentasi pada *brightness*, dilakukan secara acak dengan pencahayaan antara -15% dan +15%. Berikut ini adalah hasil dari augmentasi dari *brightness* seperti yang tertera pada Gambar 4.4.

Gambar 4. 4 Gambar Hasil Augmentasi *Brightness*

Pada Gambar 4.4, diketahui bahwa nilai *brightness* tinggi membuat gambar lebih cerah dari gambar aslinya (gambar kiri), sedangkan nilai *brightness* rendah akan membuat gambar lebih gelap dari gambar aslinya (gambar kanan).

4.3. Seleksi Model *Object Detection* YOLO

Tahap selanjutnya setelah pemrosesan dataset adalah melakukan seleksi model deteksi objek. Seleksi model deteksi objek dilakukan untuk menentukan versi terbaik dari beberapa versi model YOLO terbaru, yakni dari YOLO versi 8, YOLO versi 9, YOLO versi 10, dan YOLO versi 11. Masing-masing versi dari model YOLO memiliki 5 varian berbeda sesuai dengan jumlah parameter modelnya. Oleh sebab itu, pada penelitian ini dilakukan seleksi model dengan mengambil masing-masing varian terendah pada setiap versi model YOLO. Hal ini dilakukan dengan tujuan mengurangi waktu komputasi saat evaluasi model karena mengurangi tingkat kompleksitas saat evaluasi model. Sehingga, dipilih varian YOLOv8n, YOLOv9t, YOLOv10n, dan YOLO11n.

Proses seleksi model dilakukan dengan melakukan evaluasi model pada dataset "COCO" dan "*Mall Dataset*". Dataset COCO dilakukan sebagai tahap awal seleksi, karena dataset ini merupakan dataset yang umum digunakan untuk melakukan *benchmarking* model-model deteksi objek. Hal ini dilakukan untuk mengetahui seberapa baik performa model dalam mendeteksi objek secara umum. Sedangkan, tahap seleksi berikutnya dengan melakukan evaluasi pada dataset "*Mall Dataset*", hal ini dikarenakan dataset tersebut mengandung *frame-frame* yang menunjukkan objek khusus, yakni orang. Karena pada penelitian kali ini bertujuan untuk

mendeteksi pada rekaman pengunjung, maka dataset ini sangat sesuai dengan kebutuhan dari tujuan penelitian kali ini. Proses seleksi model kali ini dilakukan dengan menggunakan Google Colaboratory dengan menggunakan tipe *runtime* GPU T4. Berikut ini adalah penjelasan lebih lengkap mengenai tahap masing-masing evaluasi model.

4.3.1. Evaluasi pada *COCO Dataset*

Tahap pertama pada seleksi model adalah dengan melakukan evaluasi model dengan menggunakan dataset bernama COCO (*Common Objects in Context*). Setelah dilakukan evaluasi terhadap model-model, maka berikut ini adalah hasil-hasil beberapa metrik seperti yang terdapat pada Tabel 4.1.

Tabel 4. 1 Tabel Hasil Evaluasi Model pada COCO Dataset

Model	*Precision*	*Recall*	mAP50	mAP50-95	*Fitness*
YOLOv8n	0. 639	0. 536	0. 607	0. 448	0. 464
YOLOv9t	0. 679	0. 548	0. 606	0. 453	0. 468
YOLOv10n	0. 645	0. 558	0. 625	0. 462	0. 478
YOLOv11n	0. 663	0. 589	0. 671	0. 505	0. 522

Berdasarkan Tabel 4.1, didapati bahwa, nilai *precision* pada YOLOv9t mengungguli versi lainnya dengan nilai 0.679. Sedangkan, pada metrik *recall*, model YOLOv11n mengungguli versi lainnya dengan nilai 0.589. Lalu, pada metrik mAP50 dan mAP50-95, model YOLOv11n mengungguli versi lainnya dengan nilai secara berurutan sebesar 0.671 dan 0.505. Dan terakhir pada metrik *fitness*, nilai terbesar masih dimiliki oleh YOLOv11n dengan nilai 0.522. Oleh sebab itu, untuk tahap awal seleksi model, maka dipilih model YOLOv11n sebagai model deteksi objek, akan tetapi akan dilakukan tahap kedua seleksi model dengan *task* yang sesuai dengan tujuan penelitian kali ini, yakni dengan menggunakan Mall Dataset.

4.3.2. Evaluasi pada *Mall Dataset*

Setelah selesai dilakukan evaluasi tahap pertama, yakni evaluasi model-model menggunakan dataset COCO, tahap berikutnya adalah dengan mengevaluasi model-model dengan menggunakan dataset bernama "*Mall Dataset*". Hal ini dikarenakan dataset ini sesuai dengan tujuan penelitian, yakni mendeteksi dan

menghitung jumlah pengunjung. Tahap evaluasi kedua ini dilakukan dengan menggunakan Google Colaboratory pada tipe *runtime* GPU T4. Dataset ini sudah melewati tahap *preprocessing* dan augmentasi, sehingga dataset ini lebih kompleks daripada dataset orisinalnya.

Jika proses evaluasi untuk semua model sudah selesai, maka hasil metrik-metrik setiap model secara otomatis akan tersimpan dan ditampilkan. Hasil dari evaluasi model-model pada *Mall Dataset* dapat dilihat pada Tabel 4.2.

Tabel 4. 2 Tabel Hasil Evaluasi Model pada Mall Dataset

Model	*Precision*	*Recall*	mAP50	mAP50-95	*Fitness*
YOLOv8n	0.853	0.745	0.854	0.631	0.653
YOLOv9t	0.856	0.760	0.869	0.654	0.675
YOLOv10n	0.834	0.730	0.843	0.634	0.655
YOLOv11n	0.867	0.771	0.881	0.653	0.676

Berdasarkan nilai pada Tabel 4.2, didapati bahwa secara keseluruhan nilai tertinggi setiap metrik dimiliki oleh YOLOv11n. Model YOLOv11n mengungguli nilai-nilai metrik *recall*, mAP50, mAP50-95, dan *fitness* dari pada model-model lainnya, akan tetapi nilai metrik mAP50-95 masih diungguli oleh model YOLOv9t. Model YOLOv8n, YOLOv9t, YOLOv10n, dan YOLOv11n secara berurutan pada metrik *precision* adalah 0.853, 0.856, 0.834, dan 0.867. Sedangkan, model YOLOv8n, YOLOv9t, YOLOv10n, dan YOLOv11n secara berurutan pada metrik *recall* adalah 0.745, 0.760, 0.730, dan 0.771. Kemudian, model YOLOv8n, YOLOv9t, YOLOv10n, dan YOLOv11n secara berurutan pada metrik mAP50 adalah 0.854, 0.869, 0.843, dan 0.881. Kemudian, model YOLOv8n, YOLOv9t, YOLOv10n, dan YOLOv11n secara berurutan pada metrik mAP50-95 adalah 0.631, 0.654, 0.634, dan 0.653. Dan metrik terakhir, yakni *fitness*, model YOLOv8n, YOLOv9t, YOLOv10n, dan YOLOv11n secara berurutan adalah 0.653, 0.675, 0.655, dan 0. 676.

Berdasarkan hasil *benchmarking* kepada 2 dataset, yakni COCO dan Mall Dataset, didapati bahwa model YOLOv11n memiliki keunggulan jikalau dibandingkan dengan model YOLO versi 8, 9, dan 10. *Benchmarking* menggunakan varian terendah pada setiap versi model YOLO versi 8, 9, 10, dan 11, sehingga versi yang unggul akan digunakan pada penelitian kali ini. Oleh karena model

YOLOv11n mengungguli versi lainnya, maka akan dipilih seluruh varian dari model YOLO versi 11, yakni YOLOv11n, YOLOv11s, YOLOv11m, YOLOv11l, dan YOLOv11x. Hal berikutnya adalah dengan melakukan *benchmarking* untuk keseluruhan varian dari model YOLO versi 11 dengan melakukan *training*.

4.3.3. *Training* Model

Tahap berikutnya setelah mendapatkan versi terbaik dari model YOLO adalah dilakukannya proses *training* model pada seluruh varian model YOLO versi 11, yakni YOLOv11n, YOLOv11s, YOLOv11m, YOLOv11l, dan YOLOv11x. Proses *training* kali ini digunakan dengan menggunakan dataset yang sudah dipersiapkan sebelumnya pada platform Roboflow, yakni dataset yang bernama "Mall Dataset", hal ini dikarenakan "Mall Dataset" memiliki kasus yang serupa dengan penugasan pada penelitian saat ini. Proses *training* model dilakukan dengan menggunakan platform Google Colaboratory dengan spesifikasi tipe *runtime* GPU A100 dan RAM sebesar 80 GB dan VRAM sebesar 40GB. Terdapat beberapa tahapan dalam *training* model-model YOLO versi 11 ini, dimulai dari dengan menginstall beberapa *library* yang dibutuhkan, mengimpor beberapa *library*, mengambil dataset dengan API "Roboflow", memanggil model-model dari "Ultralytics", melakukan proses *training*, hingga menyimpan dan mengunduh hasil dari *training* model-model.

Setiap model dilakukan *training* dengan *epoch* sebanyak 100 kali dan *optimizer* diatur secara otomatis dengan pengaturan *default* dari *library* Ultralytics. Proses training ini dilakukan dengan tujuan untuk melakukan *update* bias dan *weigth* hingga menemukan akurasi yang meningkat dari setiap *epoch*. Setiap *epoch* dari proses *training* dilakukan dengan cara mempelajari keseluruhan dataset yang diberikan, kemudian model akan melakukan *update* parameter, seperti *weight* dan bias, dengan menggunakan *optimizer* dari pembelajaran pada *epoch* tersebut. Setelah proses *training* selesai dilakukan, maka akan terdapat nilai dari beberapa metrik deteksi objek, yakni *precision*, *recall*, mAP50, mAP50-95, lamanya waktu *training* yang dibutuhkan. Hasil *training* dari seluruh varian model YOLO versi 11 tertera pada Tabel 4.3.

Tabel 4. 3 Tabel Hasil *Training* YOLOv11

Model	*Precision*	*Recall*	mAP50	mAP50-95	Waktu *Training* (menit)
YOLO11n	0.952	0.947	0.986	0.905	39.78
YOLO11s	0.957	0.949	0.988	0.924	42.48
YOLO11m	0.959	0.946	0.987	0.928	54.48
YOLO11l	0.958	0.948	0.987	0.927	73.08
YOLO11x	0.956	0.953	0.987	0.927	90.84

Berdasarkan data pada Tabel 4.3, didapatkan bahwa metrik *precision* dari varian *nano* (n), *small* (s), *medium* (m), *large* (l), dan *extra large* (x) secara berturut-turut adalah 0.952, 0.957, 0.959, 0.958, dan 0.956 yang di mana YOLO11m (*medium*) memiliki nilai terbesar pada metrik ini. Berikutnya didapatkan bahwa metrik *recall* dari varian *nano* (n), *small* (s), *medium* (m), *large* (l), dan *extra large* (x) secara berturut-turut adalah 0.947, 0.949, 0.946, 0.948, dan 0.953, sehingga YOLO11x (*extra large*) memiliki nilai terbesar pada metrik ini. Selanjutnya didapati bahwa metrik mAP50 dari varian *nano* (n), *small* (s), *medium* (m), *large* (l), dan *extra large* (x) secara berturut-turut adalah 0.986, 0.988, 0.987, 0.987, dan 0.987, di mana YOLO11s (*small*) memiliki nilai terbesar pada metrik ini. Dan yang terakhir didapati bahwa metrik mAP50 dari varian *nano* (n), *small* (s), *medium* (m), *large* (l), dan *extra large* (x) secara berturut-turut adalah 0.905, 924, 0.928, 0.927 dan 0.927, di mana YOLO11m (*medium*) memiliki nilai terbesar pada metrik ini. Oleh sebab itu, didapati bahwa YOLO11m (*medium*) mengungguli mayoritas metrik hasil evaluasi di antara varian model lainnya, yakni unggul pada metrik *precision* dan mAP50-95. Sehingga, pada tahap seleksi model ini, untuk sementara digunakan model YOLO11m (*medium*), kemudian dilanjutkan pada tahap hasil uji skor FPS pada kasus nyata untuk menemukan model dengan performa yang baik namun memiliki kecepatan yang memungkinkan untuk digunakan.

4.3.4. Test FPS Score

Setelah diketahui bahwa model YOLO11m menjadi yang unggul karena memiliki performa terbaik secara keseluruhan di antara varian lainnya, maka hal berikutnya yang harus diperhatikan adalah tingkat kekompleksan dari model untuk

dijalankan secara *real-time* pada suatu PC. Semakin kompleks parameter pada suatu model, maka akan semakin lama suatu model akan melakukan pemrosesan gambar, baik pada saat *training*, evaluasi, ataupun deteksi secara *real-time*. Oleh sebab itu, pada penelitian kali ini dilakukan pengujian untuk mengetahui tingkat kekompleksan model pada perangkat dengan mengukur rata-rata skor FPS (*Frame per Second*) yang dihasilkan dari setiap model pada percobaan dengan menggunakan video *footage* yang sama untuk keseluruhan varian model YOLO versi 11. Pengujian skor *Frame Per Second* (FPS) berikut ini dilakukan dengan menggunakan laptop dengan spesifikasi seperti yang tertera pada Tabel 4.4.

Tabel 4. 4 Tabel Spesifikasi PC

Komponen	Spesifikasi
OS	Windows 11 64 bit
CPU	AMD Ryzen 3 3250U 2.60 GHz
GPU	AMD Radeon Graphics 2GB VRAM
RAM	12 GB
Storage	Intel SSDPEKNU512GZ

Berdasarkan Tabel 4.4 didapati bahwa spesifikasi PC yang digunakan untuk melakukan sistem pelacakan tidak menggunakan bantuan API CUDA (*Compute Unified Device Architecture*) yang memungkinkan performa *image processing* menjadi lebih cepat karena gambar dikerjakan secara paralel dengan GPU. Sehingga, karena spesifikasi PC hanya menggunakan bantuan komputasi CPU, maka akan dilakukan pengujian skor FPS pada masing-masing varian agar mendapatkan model dengan performa dan kecepatan yang seimbang untuk diaplikasikan. Hasil dari nilai rata-rata FPS pada masing-masing varian model YOLO11 dapat dilihat pada Tabel 4.5.

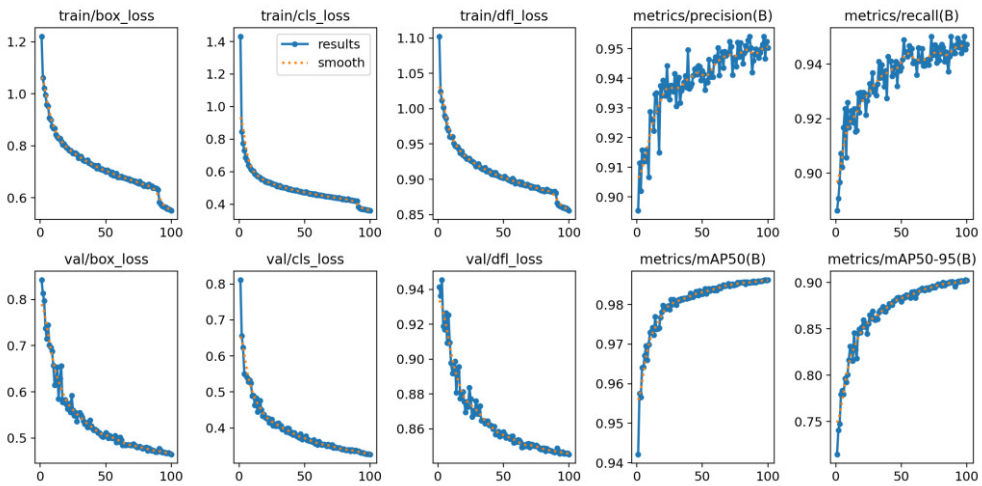
Tabel 4. 5 Tabel Hasil Skor Rata-rata FPS

Model	Rata-rata Skor FPS
YOLO11n	3.22
YOLO11s	1.53
YOLO11m	0.53

YOLO11l	0.42
YOLO11x	0.19

Berdasarkan Tabel 4.5, didapatkan bahwa model YOLO11n memiliki skor rata-rata FPS tertinggi dari varian lainnya, hal ini karena varian *nano* merupakan varian terendah dari varian lainnya pada model YOLO versi 11. Jikalau suatu video memiliki *frame* sebesar 30 FPS, berarti setiap detik pada video tersebut terdapat 30 *frame*. Maka, jikalau menggunakan YOLO11n, dibutuhkan sekitar 9.31 detik untuk menyelesaikan 1 detik video orisinalnya. Sedangkan dibutuhkan 19.60 detik untuk YOLO11s, 56 detik untuk YOLO11m, dibutuhkan 71.42 detik untuk model YOLO11l, dan dibutuhkan 157.89 detik untuk model YOLO11x.

Oleh karena kepentingan komputasi, maka digunakan versi terendah, yakni model YOLO11n. Walaupun model YOLO11n merupakan versi terendah dan paling ringan sehingga mampu memproses video dengan cepat dibandingkan dengan varian lainnya, tetapi hal ini tidak mengorbankan performa yang besar seperti hasil beberapa metrik dari proses *training* pada Tabel 4.3. Di mana metrik *precision* antara versi *nano* dan *medium* hanya berbeda 0.0007, kemudian pada metrik *recall* perbedaannya hanya 0.001, lalu pada metrik mAP50 perbedaannya hanya 0.001, dan pada metrik mAP50-95 perbedaannya hanya 0.023. Dengan perbedaan performa yang sangat kecil tersebut, model YOLO11n dapat menjalankan sistem 6 kali lebih cepat dibandingkan dengan model YOLO11m. Grafik hasil *training* dari model YOLO11n dapat dilihat pada Gambar 4.5



Gambar 4. 5 Gambar Hasil *Training* YOLO11n

Berdasarkan Gambar 4.5, terdapat 6 grafik *loss*, grafik-grafik tersebut menjelaskan seberapa besar penurunan *loss* (galat) pada model selama proses *training*, yakni 100 *epoch*. Pada grafik *box loss*, didapati bahwa terdapat penurunan eksponensial baik pada *train-set* maupun *val-set*, hal ini menandakan bahwa terjadi performa optimal pada model sehingga tidak ada *overfitting* maupun *underfitting*. Demikian juga untuk *cls_loss* dan *dfl_loss*, keduanya memiliki bentuk grafik yang sama baik pada *train-set* dan *val-set* yang terus menurun secara eksponensial. Sedangkan, 4 grafik lainnya (*precision*, *recall*, mAP50, dan mAP50-95) mengalami peningkatan secara eksponensial di setiap *epoch*-nya.

4.4. Seleksi Algoritma MoT (*Multiple-Object Tracking*)

Hal berikutnya yang dilakukan setelah selesai dengan segala urusan pada model objek deteksi adalah membandingkan algoritma pelacak (*tracker*) untuk melacak objek yang sudah terdeteksi. *Tracker* pada penelitian ini digunakan untuk melacak pergerakan objek yang sudah terdeteksi, sehingga jikalau terdapat objek dengan id yang sama pada rentang *frame* tertentu, *id* objek tersebut tidak akan terdeteksi sebagai objek yang berbeda pada *frame* berikutnya, sehingga akan tetap terhitung sebagai 1 *id* objek saja. Pada penelitian kali ini, akan dibandingkan 2 algoritma *tracker* populer, yakni BoT-SORT dan ByteTrack. Tahap pemilihan *tracker* yang terbaik dilakukan dengan *benchmarking* kedua *tracker* tersebut seperti yang akan dijelaskan di bawah berikut ini.

Oleh sebab itu, pada penelitian ini, dibandingkan kedua hasil evaluasi pada *test-set* dari hasil resmi masing-masing developer *tracker*. Pada algoritma BoT-SORT, dapat dilihat pada *link* GitHub berikut: "https://github.com/NirAharon/BoT-SORT". Sedangkan untuk algoritma ByteTrack dapat dilihat pada *link* GitHub berikut: "https://github.com/ifzhang/ByteTrack". Keduanya dievaluasi dengan menggunakan model yang sama, yakni model YOLOX. Kedua *tracker* ini juga diuji dengan menggunakan dataset yang bernama MOT20 (*Multiple-Object Tracking* 20) dan keduanya juga menggunakan *sequence* pada *test-set*. Hasil dari nilai-nilai beberapa metrik utama yang didapatkan dari kedua algoritma *tracker* pada MOT20 (*test-set*) dapat dilihat pada Tabel 4.6.

Tabel 4. 6 Tabel Hasil Evaluasi Resmi dari *Tracker*

Tracker	MOTA	HOTA	IDF1
ByteTrack	77.8	61.3	75.2
BoT-SORT	77.7	62.6	76.3

Berdasarkan Tabel 4.6, didapati bahwa kedua *tracker* memiliki performa yang hampir sama satu sama lainnya, akan tetapi BoT-SORT memiliki nilai metrik yang sedikit lebih baik dari metrik IDF1 dan HOTA. Metrik IDF1 pada BoT-SORT memiliki nilai 1.1 lebih tinggi, sedangkan pada metrik HOTA BoT-SORT memiliki nilai metrik 1.3 lebih tinggi.

Hal berikutnya setelah mengetahui nilai metrik-metrik pada studi literatur resminya, hal berikutnya yang dilakukan adalah dengan membandingkan kedua algoritma *tracker*, yakni BoT-SORT dan ByteTrack, dengan menggunakan TrackEval. TrackEval merupakan suatu program yang digunakan untuk mengetahui metrik evaluasi dari suatu *tracker*. TrackEval membutuhkan minimal 2 data untuk dibandingkan, yakni data *ground-truth* dan data hasil *tracking*. Data *ground-truth* merupakan data asli dari anotasi manual, data tersebut digunakan sebagai pembanding dengan data hasil pelacakan pada *tracker*. Pada penelitian kali ini, digunakan model YOLO11n untuk membandingkan kedua *tracker* saja, tidak untuk mendapatkan nilai metrik yang baik.

Dataset yang akan digunakan untuk mengevaluasi kedua *tracker* adalah dataset MOT-20 (*Multiple Object Tracking* 20) yang berisikan banyak *frame* dengan jumlah orang yang sangat besar. Pada proses *benchmarking tracker*, digunakan *train-set* dari dataset MOT-20 dan hanya mengambil sekuen pertama saja, yakni MOT20-01. Oleh karena dataset ini memiliki frame-frame yang sangat kompleks, maka akan sangat mungkin bahwa hasil metrik akan buruk, akan tetapi pada tahap kali ini hanya akan melihat perbandingan kedua performa dari *tracker*. Hasil dari evaluasi kedua *tracker* dengan menggunakan TrackEval dapat dilihat pada Tabel 4.7.

Tabel 4. 7 Tabel Hasil Evaluasi Dengan TrackEval

Tracker	MOTA	HOTA	IDF1
ByteTrack	17.8	32.7	36.5
BoT-SORT	17.8	32.7	36.5

Berdasarkan nilai pada Tabel 4.7, didapatkan hasil yang sangat buruk bagi keduanya, akan tetapi hal ini sangat wajar oleh karena *tracking* dijalankan hanya dengan menggunakan model seadanya yang tidak disiapkan secara khusus untuk mendeteksi kerumunan masa secara ekstrem seperti yang ada pada gambar-gambar di dataset MOT20. Akan tetapi, tujuannya adalah hanya untuk membandingkan performa pelacakan antara BoT-SORT dan ByteTrack. Maka didapatkan hasil yang sama dan tidak ada perbedaan secara signifikan dari keduanya pada dataset MOT20 *sequence* pertama (MOT20-01). Setelah membandingkannya dengan *train-set* pada MOT20, maka berikutnya adalah dengan membandingkan keduanya pada *test-set* MOT20. *Test-set* tidak dapat dijalankan secara publik, hanyalah lembaga atau organisasi tertentu saja yang terdaftar pada "motchallenge.net" saja yang dapat melakukan pengujian pada *test-set* MOT20. Berdasarkan Tabel 7 dan Tabel 8, kedua *tracker* memiliki performa yang hampir sama satu sama lainnya, akan tetapi BoT-SORT memiliki keunggulan sedikit lebih baik dari metrik IDF1 dan HOTA. Selain itu, fitur BoT-SORT juga dilengkapi dengan *Global Motion Compensation* (gmc) dan *re-ID*, sehingga membantu mengatasi pergeseran atau getaran pada video karena rekaman diambil dengan dipegang menggunakan tangan, selain itu juga membantu mengurangi galat *ID Switch* dengan bantuan *re-ID*.

4.5. *Hyperparameter* Tuning

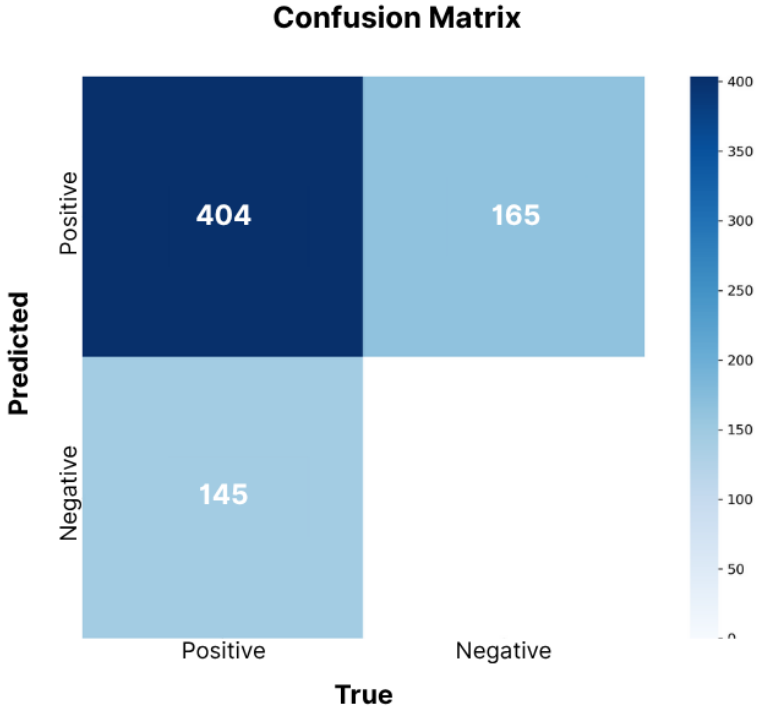
Hal berikutnya untuk memperkuat performa model deteksi adalah dengan melakukan *hyperparameter tuning*, *hyperparameter tuning* merupakan proses pengubahan beberapa konfigurasi pada model yang dapat memengaruhi kinerja model. Terdapat banyak parameter-parameter yang ada pada suatu model YOLO, pada penelitian kali ini *hyperaprameter tuning* dilakukan secara otomatis dengan menggunakan *library* Ultralytics. Dilakukan perubahan *optimizer*, yang semula adalah menggunakan optimizer dari AdamW, digantikan dengan menggunakan *optimizer* dari SGD (*Stochastic Gradient Descent*).*Hyperparamter*. Selain itu, dilakukan juga perubahan nilai learning rate, dari 0.002 menjadi 0.01, hal ini bertujuan untuk membuat proses *tuning* menjadi lebih cepat. *Tuning* kali ini dilakukan dengan menggunakan dataset gabungan dari seluruh *footage* rekaman yang diambil dengan menggunakan kamera *smartphone*. Hasil dari perbandingan

nilai metrik model sebelum dan setelah dilakukan *hyperparameter tuning* dapat dilihat Tabel 4.8.

Tabel 4. 8 Tabel Hasil Evaluasi Dengan TrackEval

Model	*Precision*	*Recall*	mAP50	mAP50-95	*Fitness*
Sebelum *Hyperparameter Tuning*	0.250	0.027	0.029	0.017	0.019
Setelah *Hyperparameter Tuning*	0.687	0.705	0.734	0.340	0.380

Berdasarkan Tabel 4.8, didapati bahwa terdapat perbedaan yang sangat signifikan dari nilai metrik-metrik seperti *Precision*, *Recall*, mAp50, mAp50-95, dan *Fitness*. Hal ini dikarenakan proses *hyperparameter tuning* ini merupakan iterasi dari *training-evaluation-logging* secara berkala hingga akhirnya menemukan parameter terbaik secara otomatis dari model untuk dataset yang spesifik. Untuk mendapatkan kesimpulan yang lebih baik, dilakukan visualisasi dengan menggunakan *Confusion Matrix*. Hasil dari *Confusion Matrix* untuk hasil evaluasi model setelah dilakukan *Hyperparameter Tuning* dapat dilihat pada Gambar 4.6.



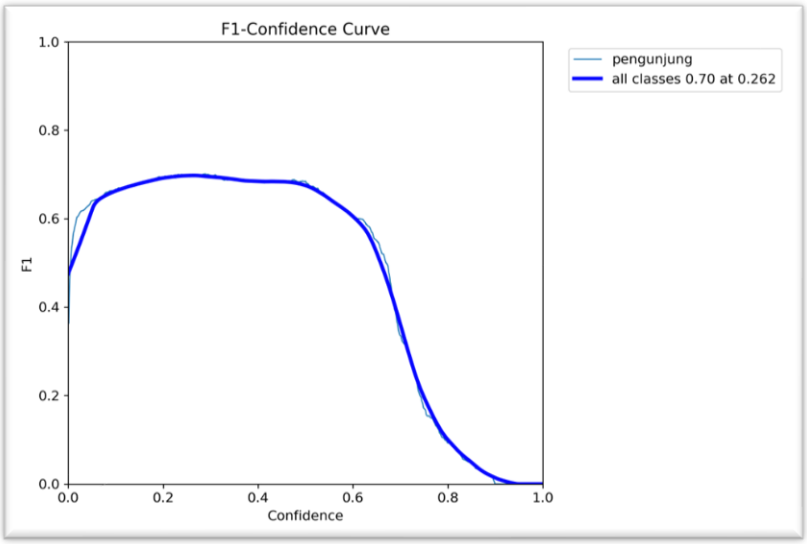
Gambar 4. 6 Gambar *Confusion Matrix* Model YOLO

Berdasarkan Gambar 4.6, terdapat suatu gambar matriks yang disebut sebagai *Confusion Matrix*. Pada *Confusion Matrix* tersebut terdapat 4 sel, yang terdiri dari *True Positive* (TP), *False Positive* (FP), *False Negative* (FN), dan *True Negative* (TN). Sel pertama pada matriks merupakan nilai dari *True Posituve* (TP), lalu sel kedua pada matriks merupakan nilai dari *False Positive* (FP), kemudian sel ketiga pada matriks merupakan nilai dari *False Negative* (FN), dan sel keempat adalah nilai dari *True Negative* (TN). Kondisi dinyatakan sebagai *True Positive* ketika model berhasil mendeteksi objek dan klasifikasi objek dengan kelas yang tepat, sedangkan kondisi *False Postive* terjadi ketika model mendeteksi namun salah dalam mengklasifikasikan kelas objek, lalu kondisi *False Negative* terjadi ketika model gagal mendeteksi objek yang seharusnya terdeteksi pada dataset *Ground Truth*, dan yang terakhir adalah kondisi *True Negative* merupakan kondisi ketika model berhasil untuk tidak mendeteksi objek-objek yang memang seharusnya tidak terdeteksi.

Hasil *Confusion Matriks* pada Gambar 4.6 mendapatkan nilai *True Positive* (TP) sebesar 404 pada sel kiri atas, yakni ketika model berhasil mendeteksi "pengunjung" ketika nilai aslinya juga "pengunjung". Setelah itu, terdapat nilai *False Positive* (FP) pada sel kanan atas sebesar 165, yakni ketika model mendeteksi objek sebagai "pengunjung" ketika seharusnya tidak ada (*background*). Kemudian, terdapat nilai *False Negative* (FN) pada sel kiri bawah, yakni ketika model tidak mendeteksi objek sebagai "pengunjung" yang seharusnya terdeteksi sebagai "pengunjung". Dan terakhir terdapat nilai *True Negative* (TN) pada sel kanan bawah, yang bernilai 0, nilai pada sel ini bertujuan untuk mengetahui seberapa banyak model tidak mendeteksi objek (*background*) yang seharusnya memang tidak terdeteksi (*background*).

Selain hasil *Confusion Matrix*, didapatkan juga hasil keharmonisan antara metrik *precision* dan *recall*, yang dinamakan sebagai metrik F1-Score. Hal ini dikarenakan metrik *precision* berbanding lurus dengan F1-score, akan tetapi metrik *recall* berbanding terbalik dengan metrik F1-Score. Hal ini menyebabkan ketika *Confidence* ditingkatkan, maka *precision* akan meningkat akan tetapi *recall* menurun, dan begitu pula sebaliknya. Maka diperlukan keharmonisan antara

precision dan *recall* yang direpresentasikan oleh nilai dari metrik F1-Score. Grafik dari F1-Score dapat dilihat pada Gambar 4.7.



Gambar 4. 7 Kurva F1-Confidence

Berdasarkan Gambar 4.7 terdapat suatu kurva yang merupakan kurva relasi antara skor *Confidence* dengan nilai F1. Skor *Confidence* merupakan skor kepercayaan model untuk mendeteksi suatu objek dengan kelas yang benar, sedangkan skor F1 merupakan skor harmoni antara *Precision* dan *Recall*. Nilai *Precision* merupakan nilai seberapa tepat model memprediksi seluruh objek yang terdeteksi dengan benar, sedangkan *Recall* merupakan nilai seberapa sering model memprediksi objek. Jikalau skor *Confidence* terlalu tinggi maka nilai *Precision* akan tinggi pula, akan tetapi nilai *Recall* akan rendah. Hal ini karena skor *Confidence* berbanding lurus dengan *Precision*, namun berbanding terbalik dengan *Recall*. Sehingga, nilai F1 pada Gambar 13 merepresentasikan nilai harmoni antara *Precision* dan *Recall* untuk menghasilkan performa deteksi yang baik. Pada hasil evaluasi, didapatkan bahwa skor *Confidence* 0.262 menghasilkan skor F1 sebesar 0.70 atau 70%.

4.6. Pengujian Kasus Nyata

Setelah sudah selesai melakukan *hyperparameter tuning* model deteksi objek, maka selanjutnya adalah mengombinasikannya dengan *tracker* yang bernama

"BoT-SORT". Sama halnya seperti model YOLO, algoritma BoT-SORT juga memiliki beberapa konfigurasi yang dapat diatur sedemikian rupa, sehingga memiliki keluasan dalam mengatur performa pelacakan berdasarkan kesulitan masing-masing kasus yang disebabkan oleh beberapa faktor, misalnya pencahayaan tempat, oklusi, ukuran objek-objek pada *frame*, dan lain-lainnya. Pada tahap ini, dilakukan konfigurasi terhadap *tracker* untuk menyesuaikan dengan model dan hasil rekaman agar dapat menghasilkan performa yang maksimal. Setiap rekaman diberikan *Region Of Interest* (ROI) untuk membatasi daerah, agar daerah tertentu saja yang dilakukan perhitungan jumlah pengunjung, tidak terhadap keseluruhan *frame* rekaman. Berikut ini adalah penjelasan lebih lengkap untuk pengujian kasus nyata.

4.6.1. Melakukan Konfigurasi *Tracker*

Tahap selanjutnya sebelum melakukan percobaan perhitungan jumlah orang adalah dengan melakukan konfigurasi pada algoritma *tracker*. Beberapa konfigurasi pada BoT-SORT dapat diubah,, seperti *track_high_thresh, track_low_thresh, new_track_thresh, track_buffer, match_thresh, proximity_thresh,* dan *appearance_thresh*. Nilai-nilai tersebut akan memengaruhi performa dari proses pelacakan, hal ini dapat diatur sedemikian rupa berdasarkan kasus-kasus tertentu, karena setiap kasus memiliki beberapa faktor yang berbeda, sehingga diberikan keluasan untuk mengatur nilai konfigurasi agar lebih maksimal dan dapat digunakan untuk kasus yang bervariasi. Nilai-nilai tersebut ditentukan secara manual pada *footage* untuk performanya pada percobaan kasus nyata. Pada penelitian kali ini dilakukan percobaan pengujian masing-masing konfigurasi secara manual pada *footage* hingga mendapatkan hasil yang terbaik berdasarkan kombinasi keseluruhan konfigurasi. Konfigurasi-konfigurasi yang digunakan pada algoritma *tracker* kali ini dapat dilihat seperti yang tertera pada Tabel 4.9.

Tabel 4. 9 Tabel Konfigurasi Algoritma *Tracker*

Konfigurasi	Nilai
Tipe *Tracker*	BoT-SORT
Track High Threshold	0.50
Track Low Threshold	0.39

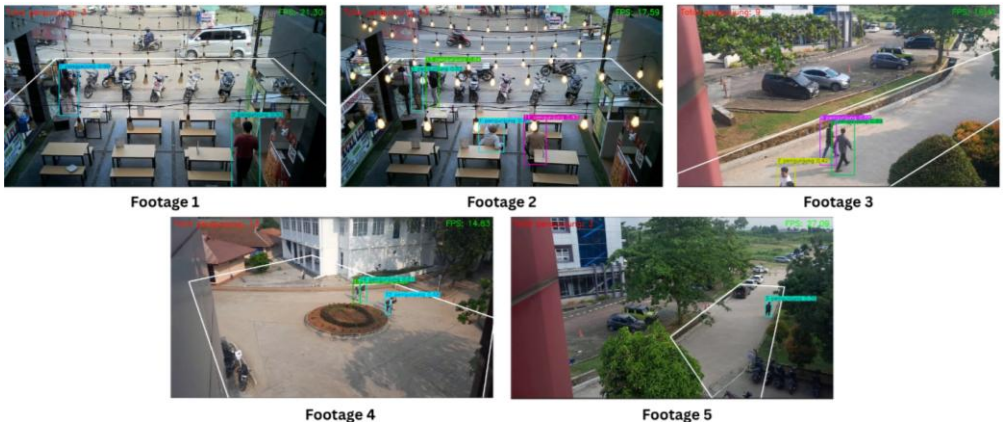
New Track Threshold	0.85
Track Buffer	900
Match Thresh	0.80
Fuse Score	True
GMC Method	sparseOptFlow
Proximity Threshold	0.60
Appearance Threshold	0.30
Re-ID	True
Model Re-ID	YOLO11n-cls

Berdasarkan Tabel 4.9, didapati bahwa nilai *Track High Threshold, Track Low Threshold, dan New Track Threshold* secara berturut-turut adalah 0.50, 0.39, dan 0.85. Ketiga nilai ini merupakan batas-batas yang dijelaskan pada Gambar 3.2, batas atas (*Track High Threshold*) merupakan nilai batas tracker untuk memberikan second association kepada identitas yang terlacak. Sedangkan rentang antara batas atas (*Track High Threshold*) dan batas bawah (*Track Low Threshold*) merupakan kondisi tracker ketika objek tersebut gagal diberikan *First Association*, sehingga *tracker* mengurangi batas agar semakin mempermudah pemberian ID pada objek tersebut. Jikalau kedua metode tersebut gagal dilakukan oleh *tracker*, maka *tracker* akan mengaktifkan pemberian ID baru yang disebut sebagai *Second Association* dengan batas *New Track Threshold*. Batas atas diatur dengan nilai 0.50 sedangkan batas bawah diberikan nilai 0.39, hal ini bertujuan agar *tracker* tidak terlalu *overfitting* ketika melacak, karena jikalau batas atas diatur terlalu tinggi, maka menyebabkan *tracker* memberikan ID yang baru secara terus-menerus kepada ID yang seharusnya tetap. Sedangkan batas bawah diberikan nilai 0.39, karena model YOLOv11 diatur dengan *Confidence Score* sebesar 0.262 seperti yang terdapat pada Gambar 4.7, akan tetapi ditingkatkan sedikit karena berdasarkan percobaan nilai terlalu rendah mengakibatkan permasalahan ID *switch*, yakni *tracker* memberikan ID yang baru secara berlebihan kepada ID yang seharusnya tetap.

4.6.2. Hasil Perhitungan Sistem

Langkah terakhir yang dilakukan pada penelitian ini adalah dengan menerapkan sistem langsung pada beberapa rekaman *footage* kasus nyata, tidak lagi

dengan menggunakan dataset seperti yang sebelumnya. Terdapat 5 rekaman yang diambil pada penelitian kali ini, kelimanya sudah merupakan campuran dari berbagai situasi dan kondisi yang berbeda. Rekaman-rekaman tersebut sudah termasuk kondisi *indoor* dan *outdoor*, serta sudah termasuk yang minim *occlusion* hingga banyak *occlusion*. Sebelum dilakukan *tracking*, kelima *footage* diberi daerah penanda untuk daerah khusus dilakukannya *tracking*, daerah tersebut dinamakan *region points* atau disebut juga sebagai *Region of Interest* (ROI). Hal ini bertujuan agar hanya daerah yang diberi ROI saja yang menghitung pengunjung dan mengabaikan objek yang berada di luar ROI. Contoh hasil penerapan RoI dan sistem penghitung pada video rekaman atau pun *real-time* yang digunakan pada penelitian kali ini dapat dilihat seperti pada Gambar 4.8.



Gambar 4. 8 Contoh Hasil Sistem Penghitung Jumlah Pengunjung

Berdasarkan Gambar 4.8, diberikan masing-masing *Region Of Interest* (ROI) secara poligon yang berbeda-beda pada masing-masing rekaman video. Hal ini dikarenakan lokasi yang berbeda, sudut yang berbeda, dan oklusi yang berbeda pada masing-masing rekaman video. Setelah dilakukan pengamatan dengan menggunakan model yang sudah dipilih, yakni YOLO11n, dan algoritma *tracker* yang sudah dikonfigurasi, maka akan dihasilkan nilai-nilai seperti yang tertera pada Tabel 4.10.

Tabel 4. 10 Tabel Hasil *Tracking* Pada Rekaman

Footage	**Total ID *Real***	**Total ID Terdeteksi**	***ID Switch***	**Total Objek Terdeteksi**	**% *Error* IDS**
Footage-1	6	11	5	6	45.45%

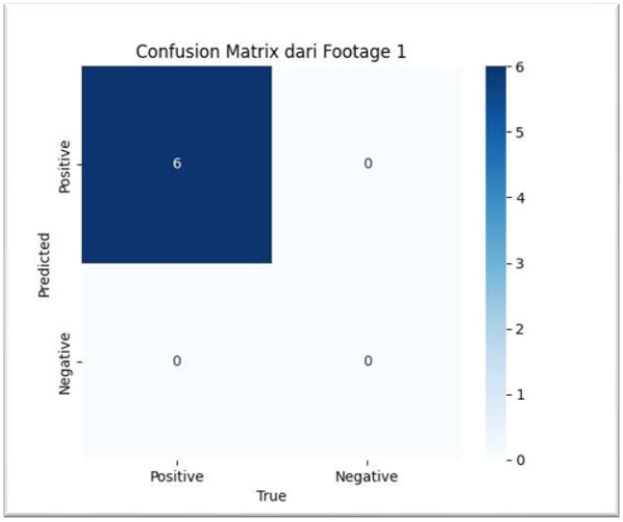
Footage-2	13	19	6	13	31.57%
Footage-3	15	11	1	10	9.09%
Footage-4	39	16	1	15	6.25%
Footage-5	19	6	1	6	16.67%

Berdasarkan Tabel 4.10, didapatkan bahwa terdapat nilai persentase *error* IDS untuk *footage* pertama, kedua, ketiga, keempat, dan kelima secara berturut-turut adalah 45.45%, 31.57%, 9.09%, 6.25%, dan 16.67%%. Nilai dari persentase *error* IDS menunjukkan seberapa besar kegagalan *tracker* untuk tetap mempertahankan ID yang tetap pada objek yang sama. Pada hasil Tabel 4.10 didapati bahwa pada *footage* pertama dan kedua terdapat persentase *error* IDS yang cukup besar yakni 45.45% pada *footage* pertama dan 31.57% pada *footage* kedua. Hal ini terjadi karena pada kedua *footage* tersebut berada pada tempat yang sama dan hanya waktu serta pencahayaan saja yang berbeda.

Lokasi pada *footage* pertama dan kedua adalah di kafe, pada rekaman tersebut terdapat banyak oklusi, yakni lampu bohlam yang digantung, sehingga sering terjadi objek pengunjung terhalang oleh lampu tersebut. Oklusi ini menyebabkan model kesulitan untuk mendeteksi pengunjung, sehingga karena model kesulitan saat mendeteksi secara konsisten pada objek, maka *tracker* juga memiliki performa yang buruk, karena setiap deteksi baru pada model akan mengakibatkan *tracker* melakukan *second association* atau pemberian ID baru meskipun terhadap orang yang sama.

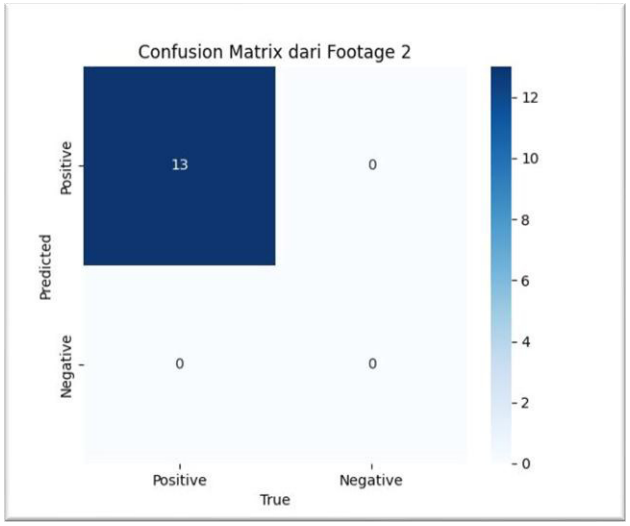
Selain mendapatkan nilai-nilai seperti yang tertera pada Tabel 4.10, untuk dapat menganalisis sistem secara lebih mendalam dapat menggunakan *Confusion Matrix*. *Confusion* Matrix berguna untuk memetakan nilai-nilai *True Positive* (TP), *False Positive* (FP), *False Negative* (FN), dan *True Negative* (TN) dalam bentuk matriks 2x2. Nilai-nilai dari TP, FP, FN, dan FN dihitung secara manual pada masing-masing rekaman video. Nilai TP diambil ketika model berhasil mendeteksi dengan benar, kemudian nilai FP didapatkan ketika model mendeteksi namun salah dalam mengklasifikasikan kelas objek. Lalu, nilai FN didapatkan ketika model gagal mendeteksi objek yang seharusnya dideteksi, dan terakhir adalah nilai TN yakni ketika model tidak mendeteksi yang seharusnya memang tidak terdeteksi.

Kemudian nilai-nilai tersebut digunakan untuk mencari metrik seperti *Precision*, *Recall*, dan skor F1. Dan berikut ini adalah *Confusion Matrix* dari masing-masing *footage* yang didapati secara perhitungan manual pada pemrosesan video secara *real-time*. Hasil dari *Confusion Matrix* untuk *footage* pertama dapat dilihat seperti pada Gambar 4.9.



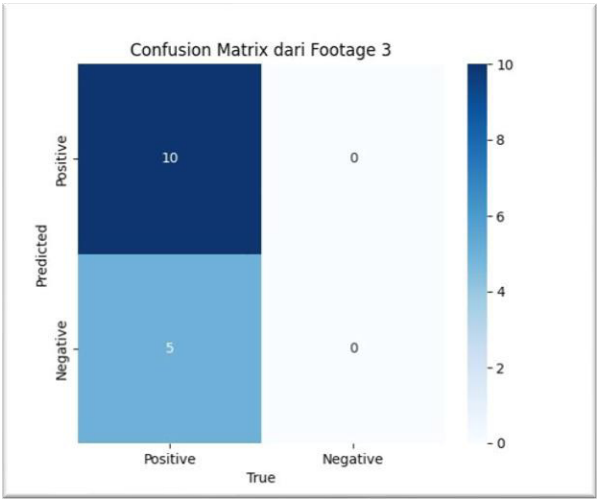
Gambar 4. 9 *Confusion Matrix* pada *Footage* 1

Berdasarkan Gambar 4.9, didapati bahwa nilai TP sebanyak 6, FP sebanyak 0, FN sebanyak 0, dan TN sebanyak 0. Maka dari itu, didapati nilai *Precision* sebesar 1 atau 100%, nilai *Recall* didapatkan sebesar 1 atau 100%. Dan skor F1 dari *footage* pertama ini adalah 1 atau 100% juga. Hasil dari *Confusion Matrix* untuk *footage* kedua dapat dilihat seperti pada Gambar 4.10.



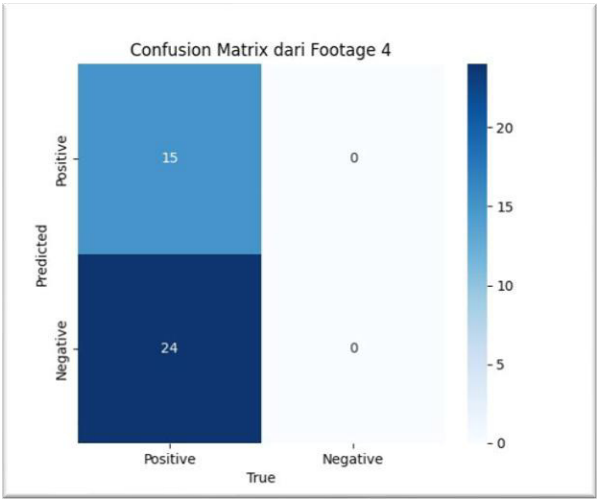
Gambar 4. 10 *Confusion Matrix* pada *Footage* 2

Berdasarkan Gambar 4.10, didapati bahwa nilai TP sebanyak 13, FP sebanyak 0, FN sebanyak 0, dan TN sebanyak 0. Maka dari itu, didapati nilai *Precision* sebesar 1 atau 100%, nilai *Recall* didapatkan sebesar 1 atau 100%. Dan skor F1 dari *footage* pertama ini adalah 1 atau 100% juga. Hasil dari *Confusion Matrix* untuk *footage* ketiga dapat dilihat seperti pada Gambar 4.11.



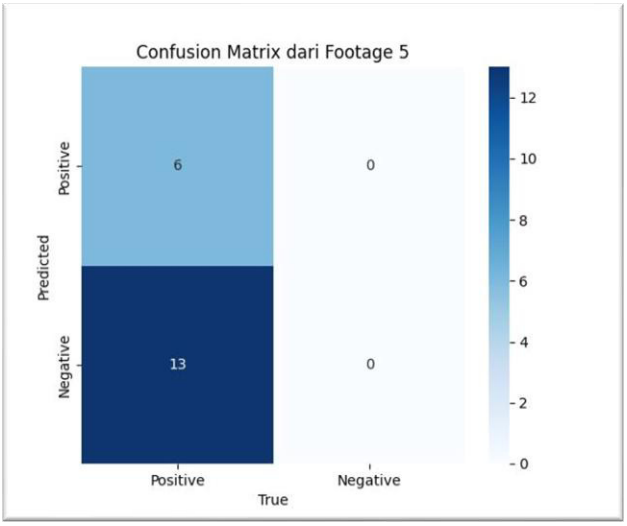
Gambar 4. 11 *Confusion Matrix* pada *Footage* 3

Berdasarkan Gambar 4.11, didapati bahwa nilai TP sebanyak 10, FP sebanyak 0, FN sebanyak 5, dan TN sebanyak 0. Maka dari itu, didapati nilai *Precision* sebesar 1 atau 100%, nilai *Recall* didapatkan sebesar 0.6667 atau 66.67%. Dan skor F1 dari *footage* pertama ini adalah 0.80 atau 80% juga. Hasil dari *Confusion Matrix* untuk *footage* keempat dapat dilihat seperti pada Gambar 4.12.



Gambar 4. 12 *Confusion Matrix* pada *Footage* 4

Berdasarkan Gambar 4.12, didapati bahwa nilai TP sebanyak 15, FP sebanyak 0, FN sebanyak 24, dan TN sebanyak 0. Maka dari itu, didapati nilai *Precision* sebesar 1 atau 100%, nilai *Recall* didapatkan sebesar 0.3846 atau 38.46%. Dan skor F1 dari *footage* pertama ini adalah 0.5556 atau 55.56% juga. Hasil dari *Confusion Matrix* untuk *footage* kelima dapat dilihat seperti pada Gambar 4.13.



Gambar 4. 13 *Confusion Matrix* pada *Footage* 5

Berdasarkan Gambar 4.13, didapati bahwa nilai TP sebanyak 6, FP sebanyak 0, FN sebanyak 13, dan TN sebanyak 0. Maka dari itu, didapati nilai *Precision* sebesar 1 atau 100%, nilai *Recall* didapatkan sebesar 0.3158 atau 31.58%. Dan skor F1 dari *footage* pertama ini adalah 0.48 atau 48% juga. Setelah dilakukan pengamatan terhadap kelima *footage*, maka didapatkan ringkasan hasil seperti yang terdapat pada Tabel 4.11.

Tabel 4. 11 Tabel Ringkasan Hasil Pengamatan Pada Rekaman

	***Footage* 1**	***Footage* 2**	***Footage* 3**	***Footage* 4**	***Footage* 5**	**Rata-rata**
% IDS	45.45%	31.57%	9.09%	6.25%	16.67%	21.81%
Precision	100%	100%	100%	100%	100%	100.00%
Recall	100%	100%	66.67%	38.64%	31.58%	67.38%
F1 Score	100%	100%	80%	55.56%	48%	76.71%

Berdasarkan Tabel 4.11, didapatkan bahwa nilai rata-rata dari seluruh *footage* pada metrik % *Error ID Switch*, *Precision*, *Recall*, dan Skor F1 secara berturut-turut adalah 21.81%, 100%, 67.38%, dan 76.71%. Nilai persentase *error ID Switch* sebesar 21.81% menandakan sistem yang dibentuk tidak memiliki banyak kesalahan pada proses pemberian ID unik kepada semua objek yang terdeteksi. Kemudian didapatkan nilai *precision* yang sangat maksimal, yakni 100%, hal ini dikarenakan model hanya diatur hanya mendeteksi satu kelas saja, yakni orang, sehingga tidak ada lagi kemungkinan model salah mendeteksi pengunjung dengan kelas lain. Sedangkan nilai metrik *recall* berhasil mencapai 67.38%, hal ini menandakan bahwa sistem berhasil membentuk *bounding box* pada orang dengan keakurasian 67.38%. Kemudian, didapatkan skor F1 sebesar 76.71% yang menandakan sistem memiliki keharmonisan antara *precision* dan *recall* sebesar 76.71%. Dengan nilai rata-rata dari seluruh metrik pada Tabel 4.11, maka didapati bahwa sistem sudah dapat dijalankan secara optimal dan berhasil mendapatkan tingkat akurasi yang sudah mencapai hasil ekspektasi.